Spencer Iannantuono
ECE 331
October 28, 2024

ECE 331 Project 1 Report

The code starts by using STP (store pair) to store the frame pointer and link register in one instruction. I subtracted 384 from the stack pointer to make room for 3 arrays of size 16 with each item having 8 bytes (3*16*8 = 384). Then I set the frame pointer to the current stack pointer. There are some commented out lines where I was manually adding arrays for testing. I then initialize the loop counter to check our array index, call that i, and counters to keep track of where we are in the prime and composite arrays. Then, the main loop begins.

The first step is to make sure that we aren't at the end of the array, which I do by using CMP (compare) to compare X3, the length (in this case, 16) with X4, the index of the main array i. If it is greater than or equal to, we branch to the end where it would return to the C code. If it doesn't branch to the end, we take the value from the A array at the correct index i using LDR, loading a[i] into X11. I then move X11 into X15 and branch with link to run the isPrime function.

In the isPrime function, we start by using MOV to initialize a value X7 as 2, we'll call that j. To know a number is prime, you can divide it by 2 and run the checks up to that number instead of the actual number. Knowing this, we use LSR to shift the value of X15 right by 1, dividing it by 2, and storing it in X8. To check if a number is prime, we begin a loop where we count up from 2 and calculate a modulo of our value of X15. If we don't find a number where the remainder is 0 before we get to the value in X8, then the value is prime.

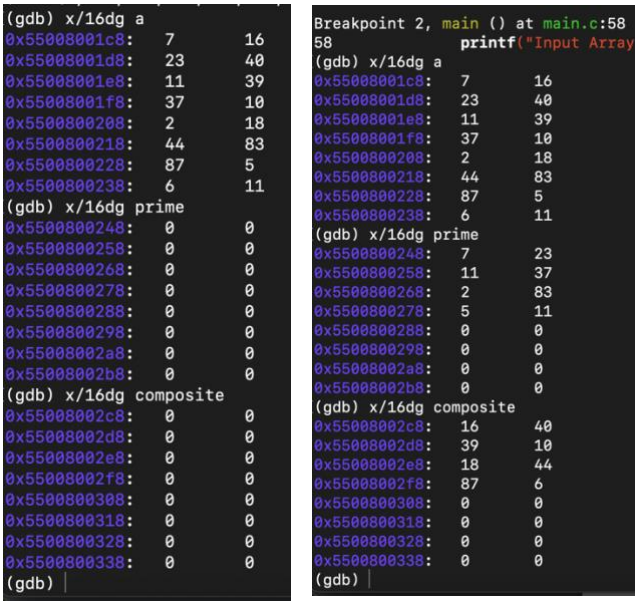
To perform a modulo operation in ARM, we use a UDIV operation (unsigned division) and MSUB operation, which subtracts a product from the value in the last register. The UDIV operation divides the number in X15 by j, and then I use MSUB to get the remainder by subtracting X9*j from the value in X15. To check if the remainder is zero, I use CBZ to branch.

If the value is zero, its composite so it branches to composite_return where I change X15 to 0 and return to the link register. If the number isn't found to be composite, I increment j by 1 and continue the loop by branching back to loop_check. If j becomes greater than X15/2, then we branch to prime_return using B.GT, change X15 to 1, and return to the link register.

When I return to the link register, I use CMP to compare X15 with 1 to see if it was prime. After the CMP instruction I use B.EQ to branch to store_prime if X15 = 1. If its prime, we branch, store the original value in X11 in the prime array, at the right index by using LSL #3. We follow the same procedure if the number is composite, but we store it in the composite array and include a B continue_loop to jump over the store_composite instructions.

The last step is to increment the value in X4 (i) by 1 in order to keep track of where we are in the input array. The program continues this process until that number has exceeded the length of the input array. As mentioned before, I use CMP to compare X4 with the length of the array, which is stored in X3. After the CMP, there is a B.GE end which will branch to the end of the program when we reach the end of the input array. Once we branch, the end block uses LDP to load a pair of values, restoring the stack and frame pointer, and then uses RET to return to the C code.

Below is a screenshot of the array memories



ARM Code

```
        .globl isPrimeAssembly
isPrimeAssembly:
        # Store FP and LR and move stack pointer 3*128
        STP X29, X30, [SP, #-384]!
        # Set FP to current SP
        MOV X29, SP

        # Initialize loop counter and indices
        # loop counter (i)
        MOV X4, #0
        # prime array idx
        MOV X5, #0
        # composite index
        MOV X6, #0

loop_start:
        # check if we are past the array length X3 = 16
        CMP X4, X3
        B.GE end

        # load a[i] into X11
        LDR X11, [X0, X4, LSL #3]

        # Move a[i] into X15 and check if its prime
        MOV X15, X11
        BL isPrime

        # Check if it was prime
        CMP X15, #1
        # if equal to 1, store in prime (with offset)
        B.EQ store_prime
```

```
# else, store in composite (with offset)
store_composite:
        STR X11, [X2, X6, LSL #3]
        # increment composite idx
        ADD X6, X6, #1
        B continue_loop

store_prime:
        STR X11, [X1, X5, LSL #3]
        # increment prime idx
        ADD X5, X5, #1

continue_loop:
        # increment i
        ADD X4, X4, #1
        # loop
        B loop_start

end:
        # Restore stack and return to C code
        LDP X29, X30, [SP], #384
        RET

isPrime:
        # X15 holds value of n
        # X7 will be j, X8 will be n / 2

        # Initialize j as 2
        MOV X7, #2

        # Only need to compare with values up to n/2, so calculate n / 2 and store in X8
        LSR X8, X15, #1
```

```
loop_check:
        # Check if j > n / 2, if it is then its prime
        CMP X7, X8

        # If j > n/2, return 1 (prime)
        B.GT prime_return

        # Perform n % j
        # If remainder is found, then it is composite

        # Quotient in X9 - n / j
        UDIV X9, X15, X7
        # remainder in X10 = n - (quotient * j)
        MSUB X10, X9, X7, X15

        # if remainder == 0, return 0 (composite)
        CBZ X10, composite_return

        # increment j by 1
        ADD X7, X7, #1

        # repeat loop
        B loop_check

composite_return:
        # Return 0 (composite)
        MOV X15, #0
        RET

prime_return:
        # Return 1 (prime)
        MOV X15, #1
```

```
RET
```